Vol.2014-ARC-211 No.15
2014/7/28

テンポラルブロッキングを適用したステンシルコードにおける階層的メモリアクセスパターン解析

松原 裕貴[1,2,a)]　佐藤 幸紀[1,2,b)]

概要：高性能計算機分野で利用されるアプリケーションのメモリアクセスは膨大である一方で、メモリアクセスの最適化を行うことはメモリウォール問題を緩和し実行時間を短縮する手段として有効である。本論文では、姫野ベンチマークにテンポラルブロッキングを適用し、その姫野ベンチマークのメモリアクセストレースを対象に我々が提案・開発を行っている階層アクセスパターン解析ツールを用いてアクセスパターンの解析を行う。階層アクセスパターン解析ツールでは、複雑なアクセスパターンが発生した場合でも検出したパターンを ID に置き換えてメモリアクセストレースとして再解釈することで再帰的にパターンの検出し、階層的にアクセスパターンを表すことが出来る。

1. はじめに

高性能計算機分野のアプリケーションの計算量は膨大であると同時に非常に多くのメモリアクセスが生じる。近年では CPU の性能向上に対するメモリの速度が追いついていないといったメモリウォール問題 [1] や高性能計算機アーキテクチャのメモリ階層構造の深化といった背景からもメモリアクセスの効率化を意識した最適化は実行時間の短縮を行う手段としてより重要なものとなると考えられる。メモリアクセスの最適化を行うためには、HPC アプリの開発者がメモリアクセスに関係する処理やデータ構造の解析を行う必要があるが、複雑かつ膨大なメモリアクセストレースからどのようなメモリアクセスが生じているかを調べるのは困難である。

このような背景のもと、我々の開発してきた実駆動型アプリケーション解析ツール Exana (Execution-driven application analysis tool）[2], [3] から得られるメモリアクセストレースを解析し、アプリ実行時に発生したメモリアクセスをメモリアクセス命令毎にストライドやシーケンシャルといったアクセスを基底パターンとして検出し、アプリ開発者にどのようなアクセスパターンが発生したかモデル化した情報を提示するアクセスパターン解析ツールの提案及び開発を行ってきた [4]。しかしながら、複雑なアクセスパターンが生じた場合は基底パターンを複数含んだ複合パターンとして検出していたため、複合パターンの行数が多い場合はアプリ開発者にとってアクセスパターンの解析が容易とは言えない状況であった。

この問題を解決するため、パターンの検出を再帰的に行うことで複雑なアクセスパターンでもモデル化が可能となる階層パターン検出手法の提案及び階層アクセスパターン解析ツールの開発を行った [5]。これにより複合パターンにストライドのような一定の規則が存在する場合、複合パターン内の基底パターンを ID、パターン間のオフセットを ID 間オフセットに変換することでメモリアクセストレースにおけるストライドと同等に解釈し、ID パターンとして検出することが可能となった。

本論文では提案した階層パターン検出手法を用いて、テンポラルブロッキング [6] を適用した姫野ベンチマークのカーネル部に関するアクセスパターンの解析を行う。テンポラルブロッキングは空間的・時間的な局所性を向上させる技術であり、キャッシュヒット率の向上が見込まれる。しかしながら、テンポラルブロッキングを適用したカーネル部は、計算範囲及びステップの階層的なループによる分割が行われるために従来より複雑なアクセスパターンが発生する。このため、階層アクセスパターン解析ツールによりブロック分割等で変化する局所性を再帰的に解析し、その結果を示す。

ここで、本論文の構成を述べる。まずメモリアクセストレースからアクセスパターンを解析する手法の概要を説明し、続けてアクセスパターンの階層的検出手法の説明を行う。次に解析解析対象となるテンポラルブロッキング版姫野ベンチマークの概要の説明を行い、実際に階層アクセスパターン解析ツールによりアクセスパターンの解析を行う。

[1] 北陸先端科学技術大学院大学 情報社会基盤研究センター
[2] JST CREST
[a)] yuuki-mt@jaist.ac.jp
[b)] yukinori@jaist.ac.jp

2. アクセスパターンの検出

本節では実駆動型アプリケーション解析ツール Exana から得られるメモリトレースからアクセスパターンを検出する手法について述べる。メモリアクセストレースは一回のメモリ参照毎の Read/write 属性、アクセスサイズ、メモリアクセス命令の番地、アクセスしたデータアドレスから構成されている。アクセスパターン解析ツールではメモリアクセストレースをメモリ参照毎に読み込み、メモリアクセス命令の番地 (以降、メモリアクセス命令とする) 毎に提案を行ったアクセスパターン解析モデルを用いてアクセスパターンの検出を行う。

図 1 にメモリアクセストレースとアクセスパターン解析ツールによって得られるアクセスパターンの一例を示す。メモリアクセストレースを順次入力し、メモリアクセス命令毎のデータアクセスの遷移に基づき検出されるパターンはフィックス、シーケンシャル、ストライド、シーケンシャルストライドの 4 つのパターンとして検出する。これら 4 つを基底パターンと呼ぶ。

図 2 に本論文で用いる基底パターンのフォーマットを示す。まず、フィックス・シーケンシャルはアクセスパターン検出モデルの先頭で表され、単体のアクセスならばフィックス、連続したアドレスへのアクセスならばシーケンシャルとなる。ストライドは一定サイズのデータアクセスとデータ間オフセットにより構成されており、データ間オフセットと後方データのストライドパートに連続性が確認される場合は連続回数 (itr) をカウントする。また、ストライドとシーケンシャルスライドの違いはアクセスされたデータ (先頭データと後方データ) がフィックスかシーケンシャルアクセスであるかの違いとなる。基底パターンのフォーマットにおいて、ストライドとシーケンシャルスライドは連続回数が 0 より大きくなる。繰り返し回数 (rep) に関しては、同一のパラメータを持つ基底パターンが連続して存在した場合、基底パターン同士の先頭アドレスが同じ場合にカウントされる。例として、二重ループによるメモリアクセスが挙げられる。

アクセスパターンの検出処理は候補パターンの構築、オフセットの検出、アクセスパターンの決定といったフェーズに分けられる。候補パターンは基底パターンが決定していない状態を指しており、トレースデータを読み込んだ際にオフセットが検出された場合、そのオフセット以前のアクセスされたデータを先頭データ (Fix/Seq) とする。検出されたオフセットをストライドパートのデータ間オフセットに設定する。この時点で、検出されたオフセット以降のデータアクセスは不明だが、ストライドの性質から先頭データと同じものを設定し、ストライドパートの繰り返し回数を 0 とすることで基底パターンのフォーマットを用いて候補パターンを構築する。続けてアクセスされたデータ

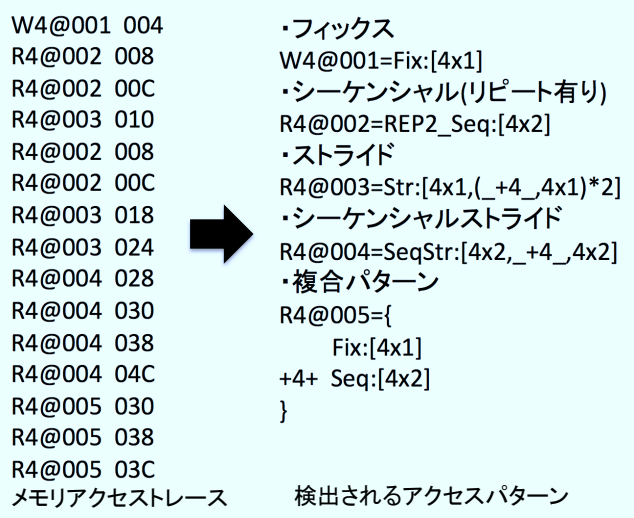

図 1 メモリアクセストレースとアクセスパターン

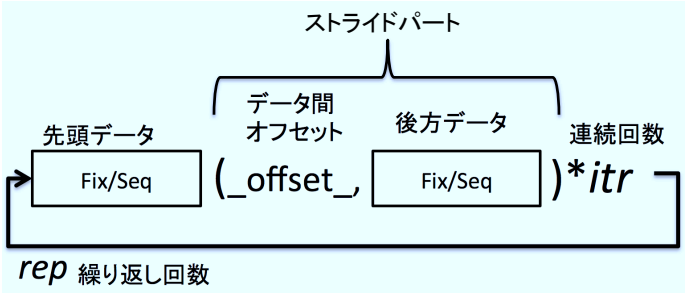

図 2 基底パターンのフォーマット

からオフセットが検出された場合、検出されたオフセットより前のデータ間オフセットと後方データが候補パターンのストライドパートと一致すれば、繰り返し回数をカウントし、どちらかが一致しない場合は候補パターンをアクセスパターンとして決定し、データ間オフセットをアクセスパターン間オフセット、後方データを先頭データ、検出されたオフセットをデータ間オフセットとした候補パターンの構築フェーズに移行する。この場合、メモリアクセス命令が複数の基底パターンを持つため複合パターンに分類される。なお、トレースデータを読み込む毎に候補パターンを利用してアクセスパターンが決定できるため、オンライン処理可能な手法となっている。

3. 階層アクセスパターンの検出手法

アクセスパターン解析ツールではアクセスパターンが一つの基底パターンにまとめられない場合、パターン間のオフセットにより複数の基底パターンを繋げた形式である複合パターンとして結果を表示する。解析対象となるアプリのメモリアクセスが大量かつ複雑な場合、複合パターンとして表示される結果はアプリの開発者にとって膨大なものとなり、アクセスパターンを理解するのは困難となる可能性が考えられる。このため、我々は複合パターン内の基底

パターンの出現に規則性が存在する場合、各基底パターンの並びからそれらの間のストライドを検知することでより柔軟にパターン化を可能とする階層アクセスパターン解析手法を提案している [5]。

階層アクセスパターンの検出では、メモリアクセスパターン解析ツールを用いて図 1 に示されるようなアクセスパターンの結果ファイルを読み込み、メモリアクセス命令毎に階層アクセスパターンの検出をオフライン処理で行う。メモリ操作命令毎に基底パターンを ID で置き換えることにより ID と ID 間オフセットによって構成される ID トレースを構築し、パターンを解析する。

階層アクセスパターンの検出処理は、基底パターン内のアクセスされたデータ部分の ID 化とデータ部分を ID に置き換えた Lv1 ID パターンの構築を行う前処理部と Lv1 ID パターンの結果を ID と ID オフセットで構築される ID トレースの変換処理及び ID トレースからストライドの検出を行い Lv2 ID パターンの構築を行う階層パターン検出処理部に分けられる。階層パターンの検出処理部では ID トレース化とストライド検出による ID パターンの構築を再帰的に行い、各処理をレベルで分けることでアクセスパターンを階層的に表現することが可能となる。

図 4 に階層アクセスパターン検出処理の流れを示す。前処理ではアクセスパターンファイルから読み込まれたメモリアクセス命令のアクセスパターンを Lv1 ID アクセスパターンに変換する。まず、基底パターンのアクセスされたデータを ID、データ間オフセットを ID 間オフセットに変換して ID パターンを構築する。アクセスパターンが複合パターンの場合、パターン間オフセットを ID パターン間オフセットにすることで Lv1 階層アクセスパターンの構築を行う。

図 3 に ID パターンのフォーマットを示す。ID パターンは基底パターンと同様のフォーマットとなっており、基底パターンのデータアクセス部分が一つ下の階層に存在する ID となり、データ間オフセットは ID 間オフセットに置き換わる。ID パターンのフォーマットと基底パターンのフォーマットの特徴的な違いは、基底パターンの場合、データアクセス部がフィックスまたはシーケンシャルアクセスに区別されていたが、ID の場合はシーケンシャルとならない。なお、ID アクセスパターン内に複数の ID パターンが含まれる場合、複合 ID アクセスパターンとする。

階層パターンの検出処理では、Lv1 階層アクセスパターンに含まれる ID パターンの ID 化と ID パターン間オフセットを ID 間オフセットに変換することで、メモリアクセストレースと同様の ID トレースへ変換を行う。ID トレースから ID 及び ID 間オフセットを読み込み、ID パターンのフォーマットと比較することで基底パターンの検出処理と同様にパターン化を行い、Lv2 ID アクセスパターンを生成する。階層パターンの検出処理では ID トレースの生成と階層アクセスパターンの検出処理を再帰的に行うことで階層的にアクセスパターンを検出する。

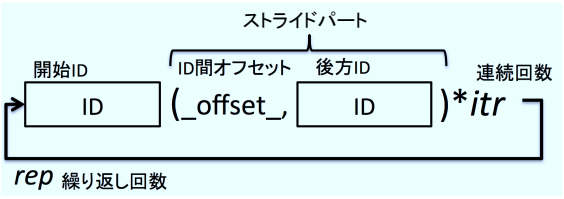

図 3　ID パターンのフォーマット

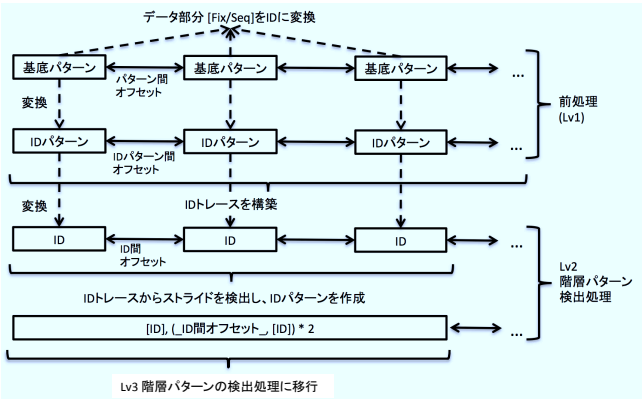

図 4　階層パターン検出の処理の流れ

4. テンポラルブロッキング版姫野ベンチ

姫野ベンチマークはポアソン方程式をヤコビ法で解くコードであり、タイムステップ分だけ x,y,z 軸からなる三次元領域においてステンシル計算を行う。x,y,z 軸の計算領域が CPU キャッシュやメインメモリに収まらない場合、計算結果が次のステップで利用される際に CPU キャッシュやメインメモリからキャッシュアウトしており再度フェッチしなければならない状況が考えられ、時間的局所性が悪い。この問題に対して、テンポラルブロッキングでは CPU キャッシュやメインメモリのサイズを考慮して計算領域を分割し、更にタイムステップも分割する。空間的に分割されたブロック内で時間的に分割されたタイムステップ分の計算を進める。これにより、分割されたブロック内で分割されたタイムステップ分計算を進めることで、一度 CPU キャッシュやメインメモリに取り込んだ計算領域を再利用可能となりキャッシュミスを減らすことが可能となる。

図 5 に、x,y 軸の計算領域を対象とするテンポラルブロッキング処理の概要を示す。x,y 軸の計算領域が全て収まるデータ領域 bufs1 と bufs2、更にスパーシャルブロックで 4 つに分割された領域の計算結果が収まる blocks1 と blocks2 を用意する。また。テンポラルブロックサイズを 3 とする。

まず計算領域内のスパーシャルブロッキングで分割された領域 スパーシャルブロック 1 を計算対象とし、全計算

領域が納められた bufs1 を入力としてステンシル計算し、blocks1 に計算結果を書き込む。なお、テンポラルブロッキングを行っているため、タイムステップを進めた際に影響を受ける領域もスパーシャルブロックから拡張 (スパーシャルブロックサイズ-1 の領域を加える) して計算を行う。次に計算結果が書き込まれたブロック 1 を計算対象として同様にステンシル計算の結果をブロック 2 に書き込む。blocks2 の計算が終えた時点で、テンポラルブロックサイズ 3 を満たすため、計算結果を bufs2 に書き込む。続けて、スパーシャルブロック 2、3、4 を同様に計算することでテンポラルブロックサイズ分の計算結果が bufs2 に書き込まれる。残りのタイムステップが存在する場合は、bufs2 を読み込み先、bufs1 を書き込み先に切り替えて同様に計算を行う。

スパーシャルブロックで分割された領域内でテンポラルブロックサイズ分計算を進めるため、読み込んだデータが再利用されるため時間的局所性が向上がする。

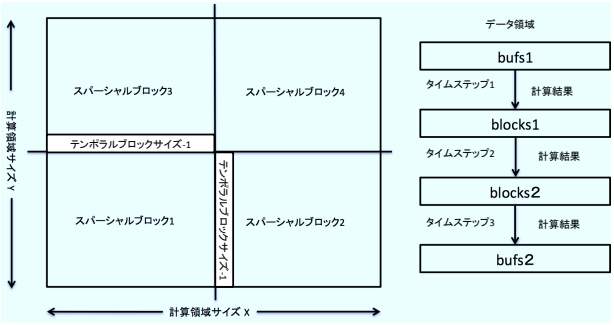

図 5 スパーシャル・テンポラルブロッキングの処理の流れとデータ領域

5. 評価実験

本節では、テンポラルブロッキングを適用した姫野ベンチマークのメモリアクセス命令を対象として、実際に実行及び取得されたメモリアクセストレースを階層アクセスパターン解析ツールによって解析した結果の一例を示し、解析結果によるアクセスパターンの説明を行う。解析結果には一部の結果のみを記載している。

表 1 にテンポラルブロッキングを適用した姫野ベンチマーク実行時のパラメータを示す。実験に用いたパラメータは姫野ベンチマークの計算領域を X,Y,Z 軸に対して二分割、ヤコビ法の総ステップ数を二分割するものである。

図 6 に主要なカーネルであるヤコビ関数の計算に関係するメモリ操作命令に着目した階層メモリアクセスパターンの解析結果の一例を示す。解析結果には、メモリアクセス命令毎にアプリ実行時に実行バイナリの ELF 情報の解析情報 (ソースコード上の関数名及び行番号、シンボル名、セクション名等) と階層メモリアクセスパターンの結果及び各レベルに含まれる ID パターンが表示される。評価実験では、シンボル名が得られなかったため階層アクセスパターンの結果の中からソースコード上の行番号と階層アクセスパターン、アクセス先のデータアドレスによってヤコビ法に関する解析対象となるメモリ操作命令を調べ、シンボル名を推定して解析を行った。

表 1 テンポラルブロッキング版姫野ベンチのパラメータ

計算領域 (X,Y,Z)	(129,65,65)
スパーシャルブロックサイズ (X,Y,Z)	(64,32,32)
ヤコビ法の総ステップ数	8
テンポラルブロックサイズ	4

```
Memory instruction name:R4@40362c
Source: jacobi_update_block
nop_fixed_ts_dynamic_himeno.c:403
Symbol: None,
R4@40362c ---Level 2 ID pattern model---
          [[ID222],(+15728+[ID222])*34]
+-39265028+[[ID482],(+1752+[ID482])*33]
+-1512232+[[ID483],(+2044+[ID483])*32]
+744188+[[ID484],(+2336+[ID484])*31]


+35869892+[[ID224],(+16768+[ID224])*32]
+-37326244+[[ID484],(+2336+[ID484])*31]
+-1476796+[[ID491],(+2628+[ID491])*30]
+779624+[[ID492],(+2920+[ID492])*29]
R4@40362c ---Level 2 Including ID Info---
[ID222]: [[ID88],(+248+[ID88])*34]
[ID482]: [[ID89],(+24+[ID89])*33]
[ID483]: [[ID90],(+28+[ID90])*32]
[ID484]: [[ID91],(+32+[ID91])*31]
[ID490]: [[ID91],(+32+[ID91])*29]
[ID224]: [[ID90],(+256+[ID90])*32]
 [ID491]: [[ID220],(+36+[ID220])*30]
[ID492]: [[ID221],(+40+[ID221])*29]
R4@40362c ---Level 1 Including ID Info---
[ID88]: 4x67
[ID89]: 4x66
[ID90]: 4x65
[ID91]: 4x64
[ID220]: 4x63
[ID221]: 4x62
```

図 6 変数 pc についての階層アクセスパターンの解析結果

図 7 にテンポラルブロッキング版姫野ベンチマークのカーネル部におけるループ階層と実行バイナリに含まれるデバッグ情報から解析対象とした命令の存在する 403 行目のソースコードを示す。ヤコビ法のループボディを駆動する x,y,z 軸方向の三重ループ、スパーシャルブロッキングの三重ループ、テンポラルブロッキングの一重ループで構成されているループボディ内部のソースコードの 403 行目の pc は bufs1,bufs2,block1,block2 をタイムステップ毎に切り替えて計算領域の結果の読み込みに利用されるポインタ変数であり、解析対象となるメモリ操作命令としては Read となっている。階層アクセスパターンの傾向が図 7 に示すループ階層と一致するため、解析対象のシンボルを pc と推定した。

表 2 にプログラム実行時の bufs1,bufs2,block1,block2 の各データ領域の開始と終了アドレスとサイズを示す。

表 2　bufs,blocks のアドレスとサイズ

	開始アドレス	終了アドレス	サイズ (byte)
bufs1	0x7fdc1689b010	0x7fdc16aaf414	2180100
bufs2	0x7fdc165e4010	0x7fdc167f8414	2180100
blocks1	0x7fdc14449010	0x7fdc144b9810	460800
blocks2	0x7fdc14337010	0x7fdc143a7810	460800

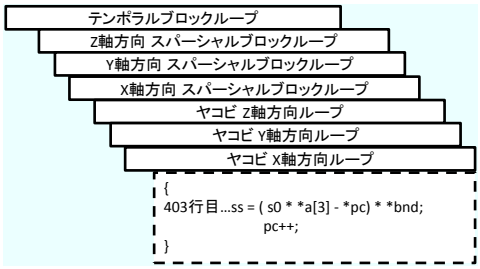

図 7　テンポラルブロッキングにおける
ヤコビ法に関するループ階層と解析対象のコード

5.1　階層アクセスパターン内の ID の解析

解析結果 (図 6) では階層アクセスパターンが複合 ID パターンとなっているため、先頭の ID パターン [[ID222],(+15728+[ID222])*34] について説明を行う。[[ID222],(+15728+[ID222])*34] は Lv2 の ID パターンであるため、要素となる ID は Lv1 の ID パターンとなる。次に ID222 の Lv1 ID パターンは [[ID88],(+248+[ID88])*34] であり、ID88 はアクセスされたデータサイズ 4*67 となっている。

図 8 に Lv1 ID パターンのアクセスパターンを示すための計算領域 (X,Y) を示す。[[ID88],(+248+[ID88])*34] の内容と合わせてアクセスパターンの解析結果の説明を行う。

テンポラルブロッキング版姫野ベンチマークでは、まず計算領域全体が含まれる bufs1 にアクセスを行う。階層アクセスパターン解析ツールでは、デバッグ用に基底パターンの開始と終了アドレスを記録しており、ID222 の開始アドレスが 0x7fdc168a351c であり、bufs1 へのアクセスであることがわかる。計算領域はスパーシャルブロッキングにより分割されているため、スパーシャルブロック 1 に対するデータアクセスを行う。

姫野ベンチマークでは計算領域の外側にアクセスされない袖領域が設けられており、また分割された領域はテンポラルブロックの処理を考慮して拡張されているため、スパーシャルブロック 1 に対する x の範囲は 1〜68 まで、y の範囲は 1〜35 までとなる。このとき、ID88 のデータアクセス範囲は 4(byte)*67 となり、スパーシャルブロック 1 の x のデータ範囲と一致する。

ID222 の ID 間オフセットは、次のデータアクセスが発生するまでのオフセットとなり計算領域の残りの x の範囲と袖領域の合計サイズが 248byte であるため ID 間オフセットの 248byte と一致する。ID222 では先頭 ID より後に ID 間オフセットと ID の組が 34 回連続して出現することが示されており、これはスパーシャルブロック 1 の y の 1 から 36 までのデータ範囲に該当する。上記の検証から ID22 は計算領域のスパーシャルブロック 1 に対するアクセスパターンを示していることがわかる。

x,y 軸方向のループが終了した際に z 軸方向に進行するため、ID222 を含む Lv2 ID パターン [[ID222],(+15728+[ID222])*34] は z 軸方向のループに対するアクセスパターンを示しており、スパーシャルブロックで分割されたヤコビ法の x,y,z 軸方向を駆動するループによって生じるアクセスパターンを ID の展開によって階層的に表すことが出来る。また、LV2 の複合 ID パターン内の ID パターンは全てヤコビ法の x,y,z 軸方向を駆動するループによって生じるものである。

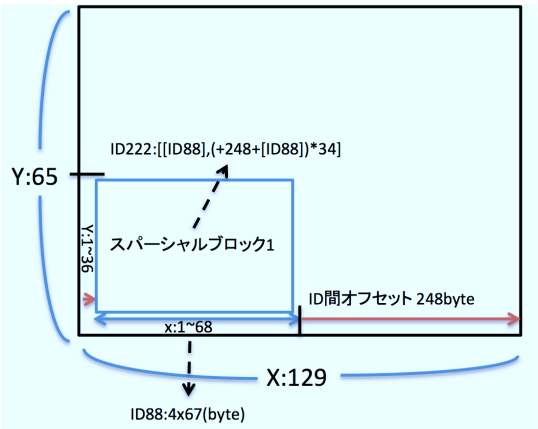

図 8　スパーシャルブロック内の ID パターンの解析

5.2　タイムステップ進行時のアクセスパターンの解析

ヤコビ法の x,y,z 軸方向を駆動するループ処理は 1 タイムステップ毎に 1 度実行される。そのため、[[ID222],(+15728+[ID222])*34] に続く Lv2 の ID パターン [[ID482],(+1752+[ID482])*33] はタイムステップが 1 進められたヤコビ法の x,y,z 軸方向を駆動するループによるアクセスパターンを示している。

テンポラルブロッキング版姫野ベンチマークではタイムステップの進行とともに計算範囲を減少させる。[ID482] を展開すると [[ID89],(+24+[ID89])*33] となり、x 軸方向のデータアクセスが 4(byte)*66 となっており、Lv2 ID パターン [[ID222],(+15728+[ID222])*34] に含まれる x,y,z 軸方向のアクセス範囲よりも 1 つずつ減少していることがわかる。同様に [[ID483],(+2044+[ID483])*32]、[[ID484],(+2336+[ID484])*31] からも計算範囲の減少が確認できる。

buf1 からの読み込み以降は読み込み先を blocks1、2 とスパーシャルブロックが変更されるまで切り替えられるため、bufs1 より読み込み先の領域が小さくなりオフセット

が減少していることがわかる。このことから、blocks をアクセス先とした処理では bufs にアクセスする場合より、空間的局所性が向上していることがわかる。

評価実験ではテンポラルブロックサイズを 4 としているため、次のスパーシャルブロック領域に移行して再びタイムステップを 1 から 4 まで処理する。全てのスパーシャルブロックが処理された場合、残りのテンポラルブロックの処理を同様に行う。なお、bufs1 が読み込み先となる場合、テンポラルブロック内の計算結果は bufs2 に記録されているため、残りのテンポラルブロックの処理のタイムステップ 1 では読み込み先が bufs2 に切り替わる。

5.3 解析結果のファイルサイズ

評価実験に用いたテンポラルブロッキング版姫野ベンチマークの全処理の解析結果のファイルサイズを表 3 に示す。メモリアクセストレースは実駆動型アプリケーション解析ツール Exana から得られるメモリアクセストレースファイル、アクセスパターンはメモリアクセストレースからアクセスパターンを解析した結果ファイル、階層アクセスパターンはアクセスパターンの解析結果から階層アクセスパターンを解析した結果であり、これらのファイルは全てテキスト形式で出力されている。なお、階層アクセスパターンの結果ファイルにはパターンやオフセット等の情報も含まれている。表 3 に示す結果から、アクセスパターンに対して階層アクセスパターンの結果ファイルサイズが約 73 %に圧縮されることがわかった。

階層アクセスパターンとして表現された形式はキャッシュ性能とアクセスパターンの関係を考察する上でも、パディングやキャッシュラインサイズを考える上で親和性が高く有用である。

メモリアクセストレース	約 5.88GB
アクセスパターン	約 3.8MB
階層アクセスパターン	約 2.8MB

表 3 解析結果のファイルサイズ

6. まとめと今後の課題

HPC アプリの実行で発生する大量かつ複雑なメモリアクセストレースから我々が研究・開発を行っている階層メモリアクセスパターン解析ツールを用いて、テンポラルブロッキングが適用された姫野ベンチマークのヤコビ法に関するメモリアクセスから階層アクセスパターンを検出し、解析を行った。

階層アクセスパターンはメモリアクセスパターンよりもメモリアクセスの特徴を抽象化することが可能となるが、複合 ID アクセスパターン内の ID パターン数の増加やメモリ領域に対する具体的なアクセスを調査するためにはアルゴリズムと照らし合わせて解析する必要があり、今後の課題として階層アクセスパターンの視覚化が必要となると考えられる。

参考文献

[1] McKee, S. A.: Reflections on the memory wall, *Proceedings of the 1st conference on Computing frontiers*, CF '04, pp. 162–167 (2004).
[2] Sato, Y., Inoguchi, Y. and Nakamura, T.: Identifying Program Loop Nesting Structures during Execution of Machine Code, IEICE Transaction on Information and Systems, Vol.E97-D, No.9, Sep. (2014 (In Printing)).
[3] Sato, Y., Inoguchi, Y. and Nakamura, T.: Whole program data dependence profiling to unveil parallel regions in the dynamic execution, *Proceedings of 2012 IEEE International Symposium on Workload Characterization (IISWC2012)*, pp. 69–80 (2012).
[4] 松原裕貴，佐藤幸紀：メモリトレース解析によるアクセスパターンのモデル化，2013 年並列/分散/協調処理に関する北九州サマーワークショップ (SWoPP 北九州 2013). 情報処理学会研究報告 2013-HPC-140，pp. 1–6 (2013).
[5] 松原裕貴，佐藤幸紀：メモリ階層の深化に対応するメモリアクセス解析ツール，ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集，p. 45 (2014).
[6] Jin, G., Endo, T. and Matsuoka, S.: A parallel optimization method for stencil computation on the domain that is bigger than memory capacity of GPUs, *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, pp. 1–8 (2013).